ANALISIS IMPLEMENTASI ALGORITMA A*(A-STAR) PADA GAME RPG(ROLE PLAYING GAME) 3D SEBAGAI DASAR PERGERAKAN NPC(NON-PLAYER CHARACTER) MENDEKATI PLAYER UNTUK MENINGKATKAN REALITAS GAME WORLD

Bonifatius Galih K[1] 22084424@students.ukdw.ac.id

Rosa Delima[2] rosadelima@ukdw.ac.id

Samuel Gandang G[3] gandang@isi.ac.id

Abstract

Game is a form of interactivity where player and game world interact to each other. In a game, one of the elements which can be considered necessary to support the course and the reality of the game is how a NPC (Non-Player Character) in the game moves. A (A-star) is an algorithm which can be used to perform pathfinding. In this case, A* will be used to find a shortest distance between the NPC and the player character. This research was performed to conduct experiments on the implementation of the A* algorithm in the 3D game. The research will be conducted by implementing an A* algorithm to a game, precisely to the enemy characters that exist in the game. Once the implementation was conducted, an experiment will performed using certain cases as a proving ground for the implementation. As a conclusion, from a hundred experiments, A* algorithm always can find the path for enemy to catch player and 73% of them are optimal paths.*

Keywords : A**algorithm*, *NPC*, *3D game*, *pathfinding*.

1. Pendahuluan

Untuk membuat sebuah *gameworld* menjadi lebih nyata dari segi cara berpindah karakter musuh, maka dibutuhkan suatu algoritma *pathfinding* yang mampu membuat karakter musuh melakukan perpindahan layaknya makhluk hidup berpindah di dunia nyata. Algoritma A*(A-Star) adalah salah satu algoritma pencarian yang dapat digunakan untuk melakukan *pathfinding*, dalam hal ini A* akan digunakan untuk mencari jarak terpendek antara karakter musuh dan karakter *player*. Jarak terpendek tersebut nantinya akan digunakan sebagai referensi arah perpindahan karakter musuh untuk mendekati *player*. Bagaimana penerapan algoritma A* dan efisiensi penerapan pada *game* yang akan dibuat adalah beberapa hal yang akan diteliti dalam penelitian ini.

2. LandasanTeori

2.1. Grafika 3D

Grafika 3D adalah grafika yang menggunakan representasi data geometri 3 dimensi yang biasanya dalam bentuk kartesian X, Y dan Z untuk merepresentasikan objek 3 Dimensi. Pada dasarnya grafika 3D masih menggunakan beberapa algoritma yang digunakan oleh grafika 2D dalam vektor dan *render*-nya(Goldstone, 2011).

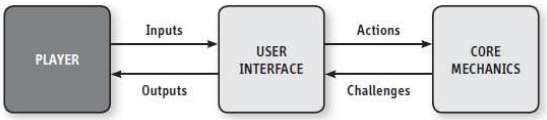
[1]*Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana*

[2]*Teknik Informatika, Fakultas Teknologi Informasi, Universitas Kristen Duta Wacana*

[3]*Animasi, Fakultas Seni Media Rekam, Institut Seni Indonesia Yogyakarta*

2.2. Game

Menurut Adams (2010) "*game* adalah suatu bentuk aktifitas bermain, bertingkah laku dalam konteks realita buatan, dimana partisipan mencoba untuk meraih minimal 1 tujuan yang penting sesuai dengan keputusan *player* dengan bertindak sesuai dengan aturan". *Game* memiliki 2 unsur yaitu *play* dan *rule*. *Play* adalah keadaan dimana player melakukan kegiatan bermain dengan *game* yang dapat membuat player menikmati *game* atau merasa terhibur dan *rule* adalah aturan yang ada dalam *game* yang menunjukkan bagaimana player bermain *game.* Hubungan *player*, *user interface* dan *core mechanics* dapat dilihat pada gambar 1. Pada gambar terlihat bahwa *user interface* merupakan komponen yang menjadi penghubung antara *player* dan *core mechanics*. Player memberikan masukan ke sistem melalui *user interface* dan selanjutnya *core machanics* memberikan keluaran berupa challenges ke *player* melalui *user interface*



*Gambar*1. Hubungan *core mechanics*, *user interface* dan *player*.
(Adams. E. (2010). Fundamentals of game design 2nd Edition.)

Terdapat beberapa unsur dalam game, beberapa unsur dalam *game* adalah *gameplay*, *game world*, *NPC*. "*Gameplay*terdiri dari tantangan yang harus dihadapi oleh *player* untuk sampai pada sasaran(*object*) dari *game* dan aksi yang boleh dilakukan oleh *player* yang ditujukan untuk tantangan tersebut"(Adams. E, 2010, hal. 11). Selain *game play*, juga terdapat *game world* yang mempengaruhi *game* menarik atau tidak untuk dimainkan.

Menurut Adam(2010),*game world* adalah "dunia buatan, dunia imajiner dimana peristiwa-peristiwa *game* berlangsung". Dunia buatan ini adalah dunia yang dapat dianggap terpisah dari dunia nyata, dimana player berperan atau berpura-pura sebagai avatar yang digunakannya.

Didalam *gameworld* terdapat karakter yang ada dalam *game*, karakter selain karakter *player* yang berjalan secara otonom juga dapat disebut dengan *NPC*(*Non-playercharacter*). NPC dibagi manjadi 3 jenis yaitu : karakter musuh, karakter kawan dan karakter pendukung.

2.3. Artificial Intelligence pada Game

Pada sebagian besar permainan, Artificial Intelligence (AI) lebih banyak digunakan untuk otomatisasi karakter pada *game*, namun sebenarnya AI juga dapat digunakan untuk membuat lingkungan yang ada dalam *game* menjadi lebih hidup. Terkait dengan hal tersebut, implementasi AI dalam sebuah permainan dapat meliputi*movement*, *decision making*, *strategy*, *infrastructure*, dan *agen-based AI*(Millington& Funge, 2009).*Movement* digunakan untuk menentukan arah pergerakan karakter secara otomatis. *Decision making* digunakan unutk menentukan tindakan yang harus dilakukan setelah karakter melakukan tindakan tertentu. *Strategy* digunakan untuk mengatur strategi dari banyak karakter sekaligus. *Infrastructure* digunakan untuk mengatur *game world* dari *game*.*Agent-based AI* adalah karakter otonom yang memproses semua informasi dalam game dan menggunakan informasi tersebut untuk mengambil keputusan.

2.4. Algoritma A*

Algoritma A* banyak diterapkan pada permainan yang terkait dengan pencari jalur/ *pathfinding*(Millington& Funge,2009). Dijelaskan bahwa A* mudah diimplementasikan, sangat efisien dan memiliki banyak bagian yang dapat dioptmalkan. Dalam algoritma A*, ruang masalah diubah menjadi *graph* dan diberikan 2 buah *node* yang akan dijadikan *start* dan *goal*. Selanjutnya *graph* yang sudah ada akan diberikan nilai yang merupakan representasi tingkat hubungan antara satu node dengan node lainnya. Berdasarkan proses pencari yang dilakukan maka akan terbentuk suatu jalur/*path* yang memiliki nilai

minimal.Jalur ini harus memiliki hubungan antara*nodestart* dan *goal*. *Path* dengan nilai minimal inilah yang nantinya merupakan *path* optimal dan akan dipilih sebagai jawaban dari masalah. Menurut Russell & Norvig(2010),untuk setiap sambungan, A* dapat menemukan *node* akhir dan menyimpan nilai total dari *path* yang sudah dilewatinya dan nilai dari *node* yang sebelumnya telah terpilih. Selain itu, A* juga menyimpan satu nilai lain, yaitu nilai estimasi dari *start* sampai *goal* menggunakan *path* yang saat itu dipilih. Nilai-nilai ini disebut nilai heuristik.

Nilai-nilai heuristik tidak boleh bernilai negatif. Pembangkitan nilai heuristik inilah yang akan menentukan apakah A* akan optimal atau tidak.Perumusan nilai heuristik dalam algoritma A* dinotasikan sebagai berikut :

= perkiraan total cost yang ditempuh dengan melalui path ini.
= cost pasti yang digunakan untuk sampai ke node dari start.
= perkiraan cost dari node ke goal.

Dari notasi-notasi tersebut, maka didapatkan perumusan sebagai berikut :

$$f(n) = g(n) + h(n) \qquad [2.1]$$

Ada 2 kondisi dimana algoritma A* dapat dianggap optimal.Pertama, kondisi *admissible* atau kondisi dimana nilai heuristic tidak berlebihan atau tidak kurang dari semestinya, salah satu contoh nilai hauristik yang dapat diterima adalah jarak lurus antara satu titik dengan titik yang lain.Kedua, *consistency* atau kondisi dimana nilai heuristik yang didapatkan dari jarak yang sudah ditempuh tidak lebih besar dari jarak yang sudah ditempuh untuk sampai node berikutnya ditambah dengan jarak lurus ke goal (Russell& Norvig(2010).

Dalam Algortima A*, penelusuran dimulai dari node awal/ *start*, selanjutnya dibangkitkan node turunannya. Node turunan akan dimasukan ke dalam daftar *open* dalam antrian. Selanjutnya dipilih node yang memiliki nilai heuristik terbaik pada antrian yang ada dan node terpilih tadi dipindahkan dari daftar *open* ke daftar *closed*. Proses terus berlanjut sampai penelusuran sampai ke node tujuan/ *goal* atau node dalam antrian kosong/habis.Setelah *node* goal ditemukan, maka *path* yang sebenarnya dicari dengan menelusur balik dari *goal* ke *start* menggunakan daftar *closed*.

2.5. Unity 3D

Unity 3D adalah salah satu alat pengembangan *game* atau biasa disebut *game engine*. Menurut Goldstone(2011) Unity 3D dikembangkan untuk membantu *game developer* agar dapat membuat *game* yang bagus dengan proses yang lebih mudah dan pengerjaan yang lebih cepat. Objek dengan banyak komponen dalam Unity 3D dapat dipecah menjadi objek-objek terpisah yang dapat diatur secara individu, hal ini dapat dilakukan karena Unity 3D menggunakan konsep *gameObject*. Komponen yang menjadi objek tersebut secara otomatis juga ditambahkan variabel yang disesuaikan dengan jenis komponen tersebut.

konsep *gameObject*pada Unity 3D meliputi*assets* atau *file* yang digunakan untuk membangun *game* dan*scene* atau *game world* dimana *game* dimainkan. Istilah *scene*digunakan karena pada dasarnya *game* menjadi dibagi menjadi beberapa *level*. Konsep lainnya adalah pada *gameObject* atau objek yang ada pada *scene* yang sedang dibuka terdapat juga *component*yang merupakan bagian dari *gameObject* untuk mengatur *gameObject*. Selain itu,terdapat *script* atau *code* yang digunakan untuk memanipulasi *gameObject*, dan *prefab* atau *gameObject* yang sudah dimanipulasi namun belum berada dalam *scene* dan disiapkan untuk setiap saat dimasukkan ke dalam *scene* selama *game* berjalan (Goldstone, 2011).

2.6. Precision and Recall

Pengukuran yang paling sering digunakan dan paling mendasar dari pencarian informasi adalah *Precision and Recall.Precision* adalah pembagian dari data relevan

diambil dari semua data yang diambil, sedangkan *recall* adalah pembagian dari data yang relevan yang diambil dari semua data relevan(Manning,Raghavan,&schütze,2008).

Tabel1.
Tabel kemungkinan Precision and Recall
(Manning, Raghavan,&Schütze, 2008).

	relevan	Tidak relevan
diambil	*True positives (tp)*	*False positives(fp)*
Tidak diambil	*False negatives(fn)*	True negatives(tn)

Tabel 1 merupakan tabel kemungkinan nilai *precision* dan *recall*, berdasarkan tabel tersebut didapatkan rumus *precision* (P) dan *recall* (R) sebagai berikut :

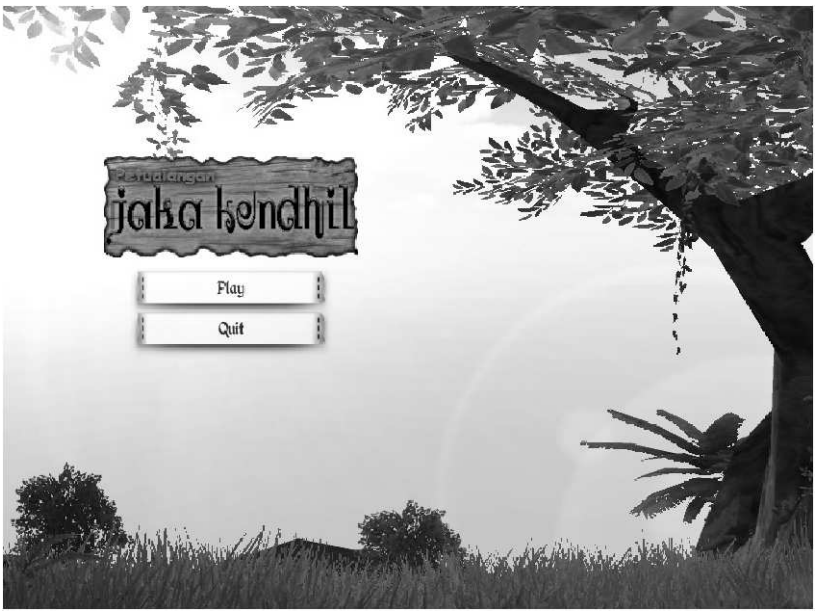
$$P = \frac{tp}{tp + fp} \qquad [2.6]$$

$$R = \frac{tp}{tp + fn} \qquad [2.7]$$

3. Hasil dan Pembahasan

3.1. Implementasi Sistem

Sistem yang telah dirancang dan digunakan sebagai alat implementasi algoritma merupakan sebuah *game* 3 dimensi yang dibuat untuk dijalankan pada *Personal Computer*(PC). Algoritma A* digunakan sebagai algoritma pencarian jarak terpendek karakter musuh untuk menemukan karakter *player*. Dalam implementasi terdapat beberapa penyesuaian dilakukan pada algoritma A*. Penyesuaian dilakukan agar algoritma dapat berjalan pada lingkungan yang ada dan memiliki hasil yang optimal.



*Gambar*2.Menu Utama *game*.

Game 3 dimensi ini dimulai dengan sebuah menu utama yang akan mengarahkan *player* masuk ke dalam *game*. Dalam menu utama terdapat beberapa pilihan menu yang dapat digunakan oleh *player*. Untuk dapat mengakses *game*, *player* dapat menggunakan menu "Play". Apabila *player* ingin keluar dari *game*, *player* dapat mengggunakan menu "Exit".

Pada layar *game*pada gambar berikut ini memperlihatkan bagaimana tampilan *game* apabila telah dimainkan. Karakter *player* akan terus berada di tengah-tengah layar karena *game* menggunakan sudut pandang orang ketiga. Titik pusat dan titik lihat dari

kamera dalam *game* adalah karakter *player* ini, sehingga dapat diputar mengelilingi karakter secara horisontal sebesar 360 derajat dan 0 derajat sampai 80 derajat secara vertikal. Saat menjalankan karakter dalam *game*, kamera akan terus berada di belakang *player*, sehingga posisi karakter akan membelakangi *player*. Gambar 2 merupakan tampilan layar dilihat dari sudut pandang pemain.

Gambar 2. Layargamedansudutpandangplayer

Algoritma A* pada implementasi ini akan bekerja apabila terpicu oleh karakter *player*. Pemicu adalah *script* untuk menghitung jarak antara karakter musuh dan karakter *player*. Gambar 3 merupakan script untuk pemicu.

```
if (Vector2.Distance(toVector2( _transform.position), toVector2( target.position)) <= Trigger)
{
    ray = new Ray(_transform.position, (target.position - _transform.position).normalized);
    jarak  =  Vector3.Distance(_transform.position, target.position);
    if (Physics.Raycast(ray, out hit, jarak))
    {
        if (hit.collider.gameObject.CompareTag("Player"))
        {
            targetwaypoint = 0;
            triggered = true;

            acceleration *= 2;
           // proximityRadius *= 1.5f;
        }
    }
```

Gambar 3. Potongan script pemicu.

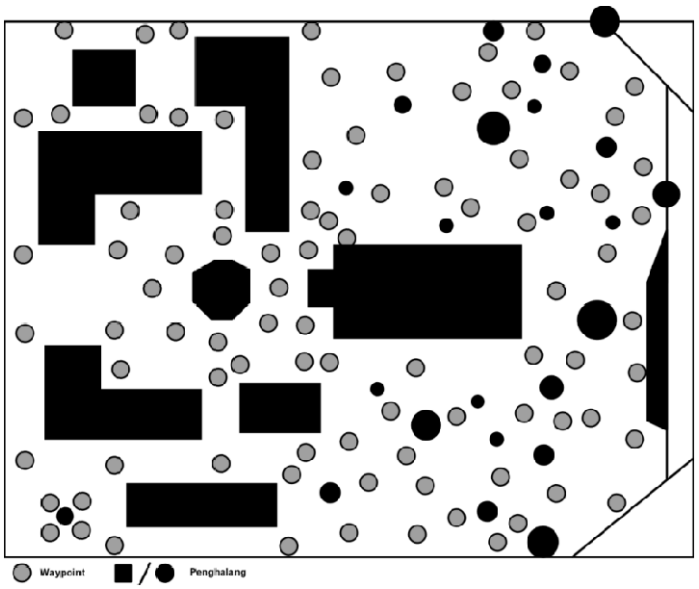
Pada saat variabel *triggered* menjadi *true*, maka selanjutnya akan dilakukan pengecekan apakah target pengejaran atau dalam hal ini adalah karakter utama terlihat oleh karakter musuh yang telah terpicu. Proses pengecekan ini juga dilakukan dengan metode *raycast*. *Script* untuk proses pengejaran dapat dilihat pada gambar 4. Apabila karakter terlihat, maka arah pergerakan karakter musuh adalah karakter *player*. Namun apabila tidak terlihat maka dilakukan pencarian jalur dengan menggunakan algoritma A*.

```
maxSpeed = maxChaseSpeed;
//targetwaypoint = 0;
isPatrol = false;
isReturn = false;
Vector3 targetPosisition = target.position;
ray = new Ray(_transform.position, (target.position - _transform.position).normalized);
jarak = Vector3.Distance(_transform.position, target.position);
if (Physics.Raycast(ray, out hit, jarak ))
{
    if (hit.collider.gameObject.tag == "Player")
    {
        viewed = true;
        if (debug) {
            Debug.DrawLine(_transform.position, target.position, Color.white);
        }
    }
    else
    {
        viewed = false;
        if(targetPosisition != lastDestination){
            Astar.instance.AStar(findClosestNode(_transform.position), findClosestNode(targetPosisition), (path) =>
            {
                if (path == null)
                {
                    return;
                }
                rute.Clear();
                rute.AddRange(path);
                CurrentWaypoints = new Vector3[rute.Count];
                rute.CopyTo(CurrentWaypoints, 0);
            });
        }
    }
}
```

Gambar 4. Potongan script pengejaran.

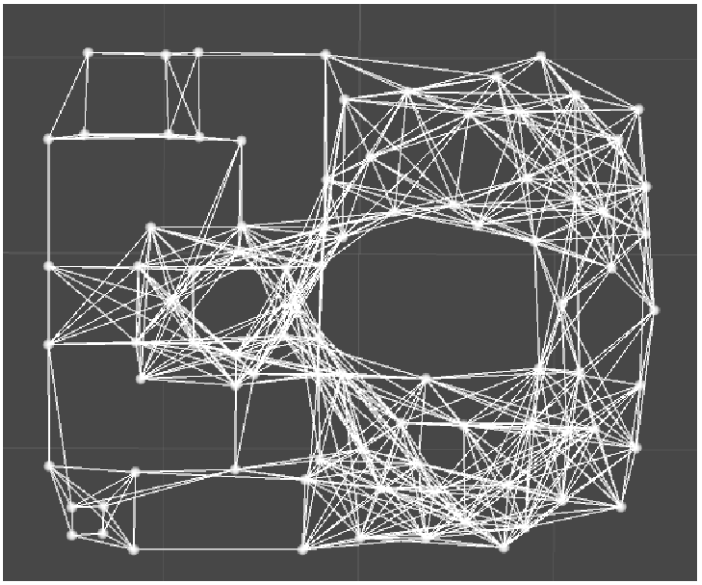
Pencarian dengan algoritma A* akan dilakukan apabila karakter *player* berpindah tempat, selama karakter *player* tidak berpindah tempat, maka pencarian dengan algoritma A* tidak akan dilakukan.

Dalam *game* ini, *node* diimplementasikan sebagai titik-titik yang merepresentasikan arah gerak yang dapat dilakukan dalam *game*. Titik-titik ini disebut dengan *waypoint*. *Waypoint* dalam *game* ini ditata dengan tujuan tertentu yaitu agar setiap *waypoint* dapat terhubung dengan *waypoint* yang lain baik secara langsung maupun secara tidak langsung. Representasi lokasi *waypoint* dapat dilihat pada gambar 5.



Gambar 5. Representasi lokasi waypoint.

Satu *waypoint* dengan yang lainnya saling berhubungan satu sama lain, sehingga apabila ditarik garis antarhubungan tersebut akan membentuk kerangka yang berbentuk seperti jaring yang berhubungan, seperti terlihat pada gambar 6. Tabel 2 merupakan daftar *waypoint* beserta *waypoint* tetangga beserta jarak antar *waypoint*.



Gambar 6. Representasi hubungan antar-waypoint dengan garis.

Tabel 2.
Contoh waypoint dengan tetangga dan jaraknya.

Waypoint	Tetangga	Jarak(Unit)
Waypoint24	Waypoint20	21,48851
Waypoint38	Waypoint37	22,49985
Waypoint47	Waypoint30	65,57852
Waypoint21	Waypoint35	70,91646
Waypoint89	Waypoint84	31,22476
Waypoint51	Waypoint59	32,32264
Waypoint21	Waypoint13	47,18161
Waypoint72	Waypoint65	49,56364
Waypoint10	Waypoint20	15,95027
Waypoint32	Waypoint51	74,58924
Waypoint32	Waypoint35	59,26629
Waypoint46	Waypoint31	70,81723
Waypoint5	Waypoint19	56,5533
Waypoint85	Waypoint65	61,38546
Waypoint57	Waypoint46	55,31351
Waypoint57	Waypoint65	44,3143
Waypoint50	Waypoint58	31,18785
Waypoint62	Waypoint79	84,21848
Waypoint1	Waypoint6	18,51481
Waypoint58	Waypoint43	78,29617
Waypoint24	Waypoint20	21,48851

3.2. Implementasi Algoritma A*

Implementasi algoritma A* dimulai dengan percobaan apakah titik asal dan titik tujuan tidak terhalang oleh obyek *game* yang ada. Apabila tidak ada obyek *game* yang menghalangi kedua titik tersebut maka titik asal dan titik tujuan dikembalikan langsung sebagai suatu jalur. Apabila terdapat penghalang yang menghalangi kedua titik tersebut maka dibuat satu buah *node* baru yang digunakan untuk memulai pencarian. Isikan posisi

dari titik awal sebagai nilai posisi dan nilai asal dari *node*. Selain itu isikan 0 untuk nilai +(nilai jumlah jarak yang sudah ditempuh) awal.

Setelah dibangkitkan *node* awal, pertama-tama dicari *node* dengan nilai , (+ + jarak lurus *node* ke titik tujuan) terkecil di dalam daftar *open*. Pencarian itu dilakukan dengan melakukan perulangan sebanyak isi dari daftar *open*. Kemudian setelah ditemukan *node* dengan nilai , terkecil disebut dengan *nodebest*, pindahkan *node* tersebut ke dalam daftar *closed* dan hapus yang ada di dalam daftar *open*.Setelah *node* dengan nilai , terbaik ditemukan dan dihapus dari daftar *open,* cari setiap tetangga dari *node best* yang belum ada dalam daftar *closed* ke dalam daftar *open* dan berikan nilai +*nodebest* ditambah dengan jarak antar *node best* dan tetangga untuk nilai + dari *node-node* baru tersebut, dan *node best* sebagai *node* awal mereka. Apabila ada tetangga yang sudah ada dalam daftar *open* dan memiliki nilai + yang lebih besar dari nilai + + jarak *node* baru ke *node best*, maka ganti nilai + dari node baru tersebut dengan + + jarak *node* baru ke *node best* dan ganti ini *node* awal dari *node* baru tersebut dengan *node best*.

Pada algoritma A* versi ini ditambahkan satu bagian khusus untuk melakukan pengujian apakah terdapat penghalang antara *node* tetangga dan target, apabila tidak terdapat penghalang, maka *node* tetangga akan dimasukkan ke dalam daftar sementara bernama daftar hasil. Setelah seluruh isi dari daftar *open* habis, maka lakukan pencarian terbalik untuk menemukan jalur yang sebenarnya. Pencarian dihentikan dengan kondisi bila ditemukan *node* awal dan *node* posisi yang sama pada satu *node*. Saat jalur ditemukan, maka kembalikan jalur sebagai hasil kembalian, apabila jalur tidak ditemukan maka kembalikan *null*.

3.3. MetodePengujian

Pengujian dilakukan dengan 2 cara, yaitu dengan mencoba di *game* dan menganalisis jalur yang dihasilkan, dan dengan menjalankan algoritma A* dengan masukan seluruh *waypoint* yang ada dalam *game*. Untuk pengujian pertama akan diberikan 8 kasus yang beragam untuk mendapatkan hasil yang berbeda-beda disesuaikan dengan kondisi *map* yang ada. Setiap kasus yang diberikan merepresentasikan masalah atau kondisi yang mungkin terjadi dalam *game* pada saat karakter musuh mengejar karakter *player*.

Pengujian kedua yaitu dengan menjalankan fungsi A* sebanyak kasus yang dapat terjadi dan mengambil 100 data hasil fungsi A* tersebut sebagai *sample* data. Dari 100 data tersebut dianalisis jumlah data yang tidak dapat menemukan hasil, data yang menemukan hasil dan optimal, data yang menemukan hasil yang tidak optimal. Dari jumlah kumpulan data tersebut dihitung nilai *precision* dan *recall*-nya untuk dapat menghasilkan kesimpulan untuk penelitian ini.

3.4. HasilPengujian

Pada pengujian pertama didapatkan bahwa untuk kasus pertama sampai ketujuh, fungsi A* dapat menghasilkan jalur yang tepat dan optimal, namun pada kasus ke delapan, fungsi A* menghasilkan jalur yang tidak optimal. Pada kasus ke delapan ini apabila yang dihitung adalah jumlah *waypoint* yang dilalui memang jalur yang dihasilkan algoritma A*(warna merah) mengunjungi *waypoint* dengan jumlah yang sama, namun jarak yang ditempuh lebih jauh dari jalur dengan garis biru.

Dari ke-delapan kasus yang ada, dapat disimpulkan bahwa algoritma A* dapat berjalan dengan baik dalam lingkungan tersebut karena dapat menemukan semua jalur, walaupun dengan beberapa kasus yang menghasilkan jalur yang tidak optimal.

Pada pengujian kedua, data yang akan digunakan dalam analisis ini merupakan 100 *sample*/contoh data yang diambil secara acak dari 7.656 data yang dihasilkan dari semua kasus yang mungkin terjadi. Semua data didapatkan dengan melakukan perulangan pada algoritma A* sebanyak jumlah *waypoint* dikalikan dengan jumlah *waypoint* - 1. Titik awal dari pencarian adalah urutan *waypoint* secara *ascending* dan titik tujuannya adalah *waypoint* secara *descending* dan apabila titik tujuan dan titik awal berada pada titik yang sama akan dilewatkan, sehingga dengan 88 *waypoint*, didapatkan 7.656(88 x 87) data. Data tersebut nantinya akan digunakan 100 data untuk analisis dengan metode *Precesion and Recall*. *Precision and Recall* dipilih karena metode ini dapat digunakan untuk menghitung tingkat

presisi suatu pencarian informasi dari suatu dokumen. Dalam analisis ini, diambil informasi dari kumpulan 7.656 data yang nantinya akan digunakan sebagai dokumen yang akan diuji tingkat presisinya.Gambar 3 adalah sebagian data yang akan digunakan sebagai dokumen yang akan diuji.

Tabel 3.
Tabel contoh data.

Asal	Tujuan	Jalur + Jarak tempuh(g)						
Waypoint16	Waypoint56	Waypoint16	17,83921	Waypoint18	167,9215	Waypoint56		
Waypoint39	Waypoint4	Waypoint39	32,56891	Waypoint28	82,40454	Waypoint30	177,592	Waypoint4
Waypoint45	Waypoint30	Waypoint45	45,77653	Waypoint40	89,39705	Waypoint30		
Waypoint25	Waypoint59	Waypoint25	31,06765	Waypoint31	70,75965	Waypoint38	156,14	Waypoint59
Waypoint11	Waypoint33	Waypoint11	88,96215	Waypoint35	177,3335	Waypoint33		
Waypoint11	Waypoint28	Waypoint11	91,86794	Waypoint28				
Waypoint81	Waypoint3	Waypoint81	41,86764	Waypoint66	115,1822	Waypoint50	304,827	Waypoint3
Waypoint73	Waypoint33	Waypoint73	69,9108	Waypoint57	149,428	Waypoint33		
Waypoint64	Waypoint2	Waypoint64	79,32909	Waypoint46	122,2133	Waypoint35	264,795	Waypoint2
Waypoint18	Waypoint53	Waypoint18	86,68533	Waypoint42	168,713	Waypoint51	194,707	Waypoint53
Waypoint70	Waypoint12	Waypoint70	239,1191	Waypoint12				
Waypoint75	Waypoint2	Waypoint75	78,61382	Waypoint53	121,8869	Waypoint45	281,678	Waypoint2
Waypoint89	Waypoint81	Waypoint89	71,01279	Waypoint87	101,7195	Waypoint81		
Waypoint61	Waypoint11	Waypoint61	76,79435	Waypoint46	119,6785	Waypoint35	208,641	Waypoint11
Waypoint63	Waypoint22	Waypoint63	46,9913	Waypoint57	226,9053	Waypoint22		
Waypoint36	Waypoint45	Waypoint36	37,30746	Waypoint38	58,91396	Waypoint45		
Waypoint88	Waypoint54	Waypoint88	84,42032	Waypoint69	131,6749	Waypoint54		
Waypoint35	Waypoint41	Waypoint35	56,81954	Waypoint38	119,1261	Waypoint40	131,654	Waypoint41
Waypoint4	Waypoint15	Waypoint4	34,57681	Waypoint15				
Waypoint69	Waypoint61	Waypoint69	61,52763	Waypoint67	212,4655	Waypoint61		

Tabel 3 di atas merupakan sebagian dari *sample* data yang akan digunakan. Dari 100 data tersebut nantinya akan dilihat apakah ada pencarian yang gagal atau tidak. Jumlah pencarian gagal atau berhasil ini digunakan sebagai nilai *relevant.* Sedangkan nilai *retrieved* didapatkan dari jumlah jalur yang optimal atau tidak dari ke 100 data yang ada. Untuk perbandingan , 100 contoh data diuji ulang secara manual untuk mengetahui apakah setiap contoh data tersebut memiliki jalur dengan jarak terpendek(optimal).

Tabel 4.
Tabel contoh data yang sudah dihitung ulang secara manual.

Asal	Tujuan	Jalur + Jarak(g)						
Waypoint16	Waypoint56	Waypoint16	17,83921	Waypoint18	167,9215	Waypoint56		
Waypoint39	Waypoint4	Waypoint39	32,56891	Waypoint28	82,40454	Waypoint30	177,5915	Waypoint4
Waypoint45	Waypoint30	Waypoint45	45,77653	Waypoint40	89,39705	Waypoint30		
Waypoint25	Waypoint59	Waypoint25	69,31651	Waypoint38	156,1344	Waypoint59		
Waypoint11	Waypoint33	Waypoint11	88,96215	Waypoint35	177,3335	Waypoint33		
Waypoint11	Waypoint28	Waypoint11	91,86794	Waypoint28				
Waypoint81	Waypoint3	Waypoint81	41,86764	Waypoint66	115,1822	Waypoint50	304,8266	Waypoint3
Waypoint73	Waypoint33	Waypoint73	69,9108	Waypoint57	149,428	Waypoint33		
Waypoint64	Waypoint2	Waypoint64	115,6159	Waypoint35	258,1972	Waypoint2		
Waypoint18	Waypoint53	Waypoint18	86,68533	Waypoint42	168,713	Waypoint51	194,7071	Waypoint53

Tabel 4.
Tabel contoh data yang sudah dihitung ulang secara manual (terusan).

Asal	Tujuan	Jalur + Jarak(g)							
Waypoint70	Waypoint12	Waypoint70	239,1191	Waypoint12					
Waypoint75	Waypoint2	Waypoint75	121,7194	Waypoint45	281,5108	Waypoint2			
Waypoint89	Waypoint81	Waypoint89	71,01279	Waypoint87	101,7195	Waypoint81			
Waypoint61	Waypoint11	Waypoint61	117,1963	Waypoint35	206,1585	Waypoint11			
Waypoint63	Waypoint22	Waypoint63	46,9913	Waypoint57	226,9053	Waypoint22			
Waypoint36	Waypoint45	Waypoint36	37,30746	Waypoint38	58,91396	Waypoint45			
Waypoint88	Waypoint54	Waypoint88	84,42032	Waypoint69	131,6749	Waypoint54			
Waypoint35	Waypoint41	Waypoint35	56,81954	Waypoint38	119,1261	Waypoint40	131,6543	Waypoint41	
Waypoint4	Waypoint15	Waypoint4	34,57681	Waypoint15					
Waypoint69	Waypoint61	Waypoint69	61,52763	Waypoint67	212,4655	Waypoint61			

Tabel 4 merupakan tabel contoh data setelah dilakukan perhitungan ulang. Pada tabel terdapat baris yang diberi warna merah, warna merah ini menunjukkan bahwa ada data dari kumpulan contoh data yang tidak optimal.

Tabel 5
Tabel kemungkinan Precision and Recall.

	Goal ditemukan	*Goal* tidak ditemukan
Optimal	73	0
Tidak Optimal	27	0

Tabel 5 adalah tabel kemungkinan dari analisis dengan metode *Precision and Recall* yang dilakukan. Dari tabel tersebut, maka diketahui bahwa nilai dari *Precision*(-) dari contoh data tersebut adalah

$$= \frac{tp}{tp + fp} = \frac{73}{73 + 0} = 1$$

Selain itu, juga terdapat nilai *Recall*(0) sebagai berikut :

$$R = \frac{tp}{tp + fn} = \frac{73}{73 + 27} = 0{,}73$$

Dengan demikian, berdasarkan contoh data dapat diketahui bahwa jalur yang dihasilkan oleh Algoritma A* pada penelitian ini memiliki kemampuan 100% akan menemukan *goal* yang diberikan dan 73% jalur yang dihasilkan oleh implementasi algoritma A* ini memiliki solusi optimal(memiliki jalur yang paling pendek).

4. Kesimpulan

Dalam penelitian ini digunakan dua uji coba untuk mengukur kinerja sistem. Pada uji coba pertama didapatkan bahwa Algoritma A* dapat diimplementasikan pada *game* 3 Dimensi sebagai algoritma pencarian jejak karakter musuh saat mengejar karakter player dengan memberikan pemicu.Pemicu digunakan untuk penggerak agar karakter musuh memanggil fungsi A* dan menggunakan *waypoint* sebagai pengganti *node* dari algoritma A* ini.

Berdasarkan uji coba kedua, dapat disimpulkan bahwa selama *game* dengan implementasi algoritma A* ini selalu dapat memberikan solusi jalur untuk karakter musuh mendekati karakter *player* saat mengejar dimanapun kedua karakter tersebut berada selama

karakter musuh masih terpicu, dan 73% dari jalur yang dihasilkan merupakan jalur optimal (memiliki jarak terpendek).

Daftar Pustaka

Adams, E. (2010). *Fundamentals of Game Design 2nd Edition*. Barkeley, California : New

Goldstone, W. (2011). *Unity 3.X Game Development Essentials*. Birmingham : Packt Publishing.

Manning, C . D., Raghavan, P. ,&Schütze, H., (2008). *Introduction to Information Retrieval* . Cambridge : Cambridge University Press.

Merrick, K . E. & Maher, M . L., (2009). *Motivated Reindorced Learning : Curious Chacaters for Multiuser Games*. Springer.

Millington, I .& Funge, J .. (2009) . *Artificial Intelligence for Games 2nd Edition.* Massachusetts : Morgan Kaufmann.

Reynolds, C. W. (1999) . *Steeringbehaviors for autonomous characters* . Diakses tanggal 19 Juni 2012 dari https://docs.google.com/viewer?a=v&q=cache:QFzlPxpPmYMJ :citeseerx.ist.psu.edu/viewdoc/download%3Fdoi%3D10.1.1.86.1589%26rep%3Drep1%26type%3Dpdf+&hl=id&pid=bl&srcid=ADGEESjHPMw5PORvnvPFtLmftjB44sEaOSf1W_wji18cFeI54Evew5Xj3G3LYrot9bDTT3RUUIvbr3Ofzfgv6BAy7D0vX0DfVPAj6jv9FR00nmnw3wIby0_EKFDzT-AvnGrq-BQ_J1JI&sig=AHIEtbQXIqcKjzlZ8bDOPfxe9pJRSTdurQ.

Russel, S. & Norvig, P. (2009). *Artificial Intelligence : A Modern Approach 3rd Edit ion*. New Jersey : Prentice Hall.